Solving Graph Problems via Potential Maximal Cliques : An Experimental Evaluation of the Bouchitté-Todinca Algorithm

Korhonen, Tuukka

2019-06

Korhonen , T , Berg , J & Järvisalo , M 2019 , ' Solving Graph Problems via Potential Maximal Cliques : An Experimental Evaluation of the Bouchitté-Todinca Algorithm ' , ACM Journal of Experimental Algorithmics , vol. 24 , no. 1 , 1.9 . https://doi.org/10.1145/3301297

http://hdl.handle.net/10138/309131
https://doi.org/10.1145/3301297

acceptedVersion

Solving Graph Problems via Potential Maximal Cliques: An Experimental Evaluation of the Bouchitté–Todinca Algorithm

TUUKKA KORHONEN, JEREMIAS BERG, and MATTI JÄRVISALO, University of Helsinki, Finland

The BT algorithm of Bouchitté and Todinca based on enumerating potential maximal cliques, originally proposed for the treewidth and minimum fill-in problems, yields improved exact exponential-time algorithms for various graph optimization problems related to optimal triangulations. While the BT algorithm has received significant attention in terms of theoretical analysis, less attention has been paid on engineering efficient implementations of the algorithm for different problems and thereby on empirical studies on its effectiveness in practice. In this work, we provide an experimental evaluation of an implementation of the BT algorithm, based on our second place winning entry in the 2nd Parameterized Algorithms and Computational Experiments Challenge (PACE 2017), extended to several related graph problems: treewidth, minimum fill-in, generalized and fractional hypertreewidth, and the total table size problem. Instead of focusing on problem-specific optimization of BT for a particular problem, our focus in this work is on studying the applicability of BT more generally to a range of problems. Based on the results, we conclude that an efficient implementation of the BT algorithm yields an empirically competitive approach to each of the considered problems when compared to available implementations of alternative problem-specific algorithmic approaches.

CCS Concepts: • **Theory of computation → Algorithm design techniques**; **Dynamic programming**; **Theory and algorithms for application domains**; *Graph algorithms analysis.*

Additional Key Words and Phrases: Potential maximal cliques, Bouchitté-Todinca algorithm, empirical evaluation, treewith, generalized hypertreewidth, fractional hypertreewidth, minimum fill-in, chordal completion, total table size

ACM Reference Format:
Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. 2019. Solving Graph Problems via Potential Maximal Cliques: An Experimental Evaluation of the Bouchitté–Todinca Algorithm. *ACM J. Exp. Algor.* 24, 1, Article 1.9 (February 2019), 19 pages. https://doi.org/10.1145/3301297

1 INTRODUCTION

Enumeration of potential maximal cliques yields improved exact exponential-time algorithms for various graph optimization problems where the optimal solutions are related to optimal graph triangulations with respect to different cost functions. A central algorithm in this setting is the BT algorithm first proposed by Bouchitté and Todinca for the treewidth and minimum fill-in problems [16]. The BT algorithm for treewidth and minimum fill-in enumerates potential maximal cliques in order to solve the problems via dynamic programming. The algorithm runs in polynomial

Authors' address: Tuukka Korhonen, tuukka.m.korhonen@helsinki.fi; Jeremias Berg, jeremias.berg@helsinki.fi; Matti Järvisalo, matti.jarvisalo@helsinki.fi, Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland.

1084-6654/2019/2-ART1.9 $15.00
https://doi.org/10.1145/3301297

time with respect to the number of potential maximal cliques, or more precisely, with respect to the number of minimal separators of the input graph. Later on, the BT algorithm has received significant attention in terms of theoretical analysis. It has been shown to be applicable to a range of cost functions [11, 24, 27] and to yield new exact algorithms and improved worst-case upper bounds [23, 25, 34]. However, from a more experimental perspective, less attention has been paid on engineering efficient implementations of variants of the BT algorithm for different problems and on empirical studies on the effectiveness of the implementations. As Bodlaender and Fomin put it in the context of treewidth, "While these algorithms provide the best-known running times, they are based on computations of potential maximal cliques and may be difficult to implement" [12]; the lack of empirical analysis may be due to the difficulty of engineering efficient implementations of the approach.

As a recent development, the best-performing top-3 algorithm implementations in the exact minimum fill-in track of the 2nd Parameterized Algorithms and Computational Experiments challenge (PACE) [22] were based on adaptations the BT algorithm for the minimum fill-in problem. In this work, we provide an experimental evaluation of our implementation of the BT algorithm, based on our second place winning entry in the PACE challenge, extended to several related graph problems. Instead of focusing on problem-specific optimization of BT for a particular problem, such as the so-called positive instance driven dynamic programming for treewidth (PIDDT) [52] BT approach specific to treewidth, our focus in this work is on studying the applicability of BT more generally to a range of problems. For each of the problems, we report results from an experimental comparison of the empirical performance of our open-source BT algorithm implementation with available—to the best of our knowledge state-of-the-art—implementations of other algorithmic approaches proposed for the particular problem. In particular, we consider the following five problems.

Treewidth [23, 27, 46], intuitively giving a measure for how close a graph is to a tree, is a central notion in the analysis of tractable fragments of NP-hard problems [8, 9]. For example, constraint satisfaction problems and Bayesian inference [18, 20, 26] are exponential only in the treewidth of the underlying graph representations of instances, yielding tractability for classes of instances with bounded treewidth. We compare our BT implementation for treewidth to both the earlier-proposed QuickBB branch-and-bound algorithm [30] and a declarative approach based on encoding the problem as maximum satisfiability (MaxSAT) [3] and applying a state-of-the-art MaxSAT solver to solve the resulting MaxSAT instance. Furthermore, we report results for PIDDT [52], a variant of BT with treewidth-specific optimizations.

Minimum Fill-In [23, 27, 48] is an alternative definition of optimal triangulations as those which are obtained by minimizing the number of added edges to a given graph. Interchangeably referred to as the chordal completion problem [5, 13], minimum fill-in has applications in e.g. Gaussian elimination for sparse matrices [49] and phylogenetics [34, 35, 38]. We compare our BT implementation to a recently-proposed integer programming approach [4] that uses lazy constraint generation and a heuristic separation method specifically designed for the minimum fill-in problem.

Generalized and Fractional Hypertreewidth [31, 33, 45] are generalizations of the central notion of treewidth to hypertrees, with motivations in refined analysis of constraint satisfaction problems [31, 33, 44]. Here we compare our BT implementation to the BB-ghw algorithm [51], a specialized branch-and-bound approach to generalized hypertreewidth, as well as to the det-k-decomp backtracking algorithm [32] specific to (non-generalized) hypertreewidth.

Total Table Size [11, 37, 41, 47], compared to treewidth and minimum fill-in, gives more exact bounds on the memory and time requirements for inference over Bayesian networks [39, 40]. More specifically, an optimal triangulation in terms of total table size minimizes the sum of the sizes of the conditional probability tables of a given Bayesian network, providing best performance guarantees for the junction tree inference algorithm [39, 40]. We compare our BT implementation to the recently proposed EDFS (extended depth-first search) branch-and-bound algorithm [41].

Based on the results, we conclude that an efficient implementation of the BT algorithm yields an empirically competitive approach for each of the considered problems.

The rest of this article is organized as follows. We start with necessary preliminaries on graph-related concepts and terminology (Section 2). Then, we overview the types of problems and cost functions to which the BT algorithm can be adapted to (Section 3), and give a description of the generic BT algorithm (Section 4). Further implementation details and main results from the experimental evaluation are presented in Section 5.

2 PRELIMINARIES

We recall graph-related concepts to the extent necessary for the remainder of this paper. We assume graphs to be undirected and simple.

The set of vertices and edges of a graph G are denoted by $V(G)$ and $E(G)$, respectively. A graph G is complete if it has an edge between every pair of vertices. Given a set of vertices S we denote the set of edges of a complete graph having S as vertices by S^2, i.e., $S^2 = \{\{u, v\} \mid u, v \in S, u \neq v\}$. Given a subset $S \subset V(G)$, the graph $G[S]$ induced by S has $V(G[S]) = S$ and $E(G[S]) = E(G) \cap S^2$. To simplify notation, let $G \setminus S = G[V(G) \setminus S]$. The neighborhood $\mathrm{N}(v)$ of a vertex $v \in V(G)$ contains the nodes u for which $\{u, v\} \in E(G)$, i.e., $\mathrm{N}(v) = \{u \mid \{u, v\} \in E(G)\}$. This notation is extended to a set S of vertices by $\mathrm{N}(S) = \cup_{v \in S} \mathrm{N}(v) \setminus S$.

A graph G is chordal if every cycle of length at least 4 has a chord, i.e., an edge joining two non-adjacent vertices in the cycle. A *triangulation* H of G is a chordal graph that contains G, i.e., such that $V(H) = V(G)$ and $E(G) \subset E(H)$. A triangulation H of G is minimal if no proper subgraph of H is a triangulation of G. We denote the set of minimal triangulations of G by $\mathrm{MT}(G)$. The edges in $E(H) \setminus E(G)$ are *fill edges*, denoted in set-notation by $\mathrm{FE}_H(G)$, dropping the subscript when clear from context.

A subset $\omega \subset V(G)$ is a *clique* if $G[\omega]$ is complete. A clique ω is maximal if no other clique ω' satisfies $\omega \subsetneq \omega'$. We denote the set of maximal cliques of G by $\mathrm{MC}(G)$. A set of vertices $\Omega \subset V(G)$ is a potential maximal clique (PMC) if there is a minimal triangulation $H \in \mathrm{MT}(G)$ with $\Omega \in \mathrm{MC}(H)$. We denote the set of all potential maximal cliques of G by $\Pi(G)$.

Example 2.1. Consider the example graphs in Figure 1. Starting with the graph G on the left, the set $\{v_2, v_3, v_4\} \in \mathrm{MC}(G)$ is an example of a maximal clique of G. For this graph $\mathrm{N}(v_5) = \{v_3, v_6\}$

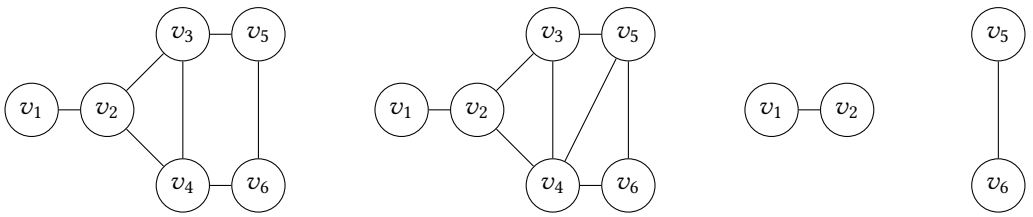

Fig. 1. Example graph G (left), a triangulation H of G (middle), and an induced subgraph $G \setminus \{v_3, v_4\}$ (right).

and $\mathrm{N}(\{v_5, v_6\}) = \{v_3, v_4\}$. The graph G is not chordal as witnessed by the cycle (v_3, v_4, v_6, v_5). An example of a minimal triangulation $H \in \mathrm{MT}(G)$ is the graph in Figure 1 middle. For this triangulation $\mathrm{FE}_H(G) = \{\{v_4, v_5\}\}$, we can also see for example that $\{v_3, v_4, v_5\} \in \Pi(G)$. Finally, the graph $G \setminus \{v_3, v_4\}$ is shown in Figure 1 right.

Two vertices u and v of G are connected if there is a path between them. A set $C \subset V(G)$ is a connected component of G if any two vertices $u, v \in C$ are connected in $G[C]$ and no nodes in C are connected to any nodes in $V(G) \setminus C$ in G. We denote the set of connected components of G by $\mathcal{C}(G)$. A set $S \subset V(G)$ is a separator of G if the graph $G \setminus S$ has at least two connected components. For a separator S, a component $C \in \mathcal{C}(G \setminus S)$ is a full component of S if $\mathrm{N}(C) = S$. A separator S is minimal if it has at least two full components. We denote the set of minimal separators of G by $\Delta(G)$. A tuple (S, C) consisting of a minimal separator $S \in \Delta(G)$ and a component $C \in \mathcal{C}(G \setminus S)$ of S is a block associated to S. A block (S, C) is full if C is a full component of S. In this work we only consider full blocks, and will for simplicity use the term block to refer to full blocks. Whenever clear from context, we also use (S, C) to denote the nodes in $S \cup C$. The realization $R(S, C)$ of a block (S, C) is the graph with $V(R(S, C)) = S \cup C$ and $E(R(S, C)) = E(G[S \cup C]) \cup S^2$. In words, $R(S, C)$ is obtained from $G[S \cup C]$ by completing S into a clique. A block (S, C) is associated with a potential maximal clique Ω if $C \in \mathcal{C}(G \setminus \Omega)$ and $\mathrm{N}(C) = S$. Notice that this also implies $S \subset \Omega$. Given a block and a PMC Ω the set $(S, C : \Omega)$ contains all blocks (S_i, C_i) associated with Ω for which $(S_i, C_i) \subset (S, C)$.

Example 2.2. Let $S = \{v_3, v_4\}$ and consider the graphs G and $G \setminus S$ in Figure 1 left and right, respectively. As G is connected, we have $\mathcal{C}(G) = \{V(G)\}$. Similarly $\mathcal{C}(G \setminus S) = \{\{v_1, v_2\}, \{v_5, v_6\}\}$. Hence S is a separator of G. As both $(S, \{v_5, v_6\})$ and $(S, \{v_1, v_2\})$ are blocks associated with S, S is a minimal separator. The block $(S, \{v_1, v_2\})$ is associated with the PMC $\{v_3, v_4, v_5\}$.

3 PROBLEMS COMPUTABLE USING THE BT ALGORITHM

We overview three classes of objective functions which can be addressed with the BT algorithm, based on the abstract framework of [11, 27]. As instantiations of the framework, we provide definitions for the five concrete optimization problems considered in this paper.

Given an undirected graph, the BT algorithm can be used for determining the value of $f(G)$ for different types of graph parameters f. Specifically, the BT algorithm is applicable whenever the function f is one of the following three types.

Clique-type [27]: f is of clique-type if

$$f(G) = \min_{H \in \mathrm{MT}(G)} \max_{\omega \in \mathrm{MC}(H)} g_{\mathrm{c}}(\omega)$$

for some clique-function $g_{\mathrm{c}} \colon 2^{V(G)} \to \mathbb{R}^+$.

Fill-type [27]: f is of fill-type if

$$f(G) = \min_{H \in \mathrm{MT}(G)} \sum_{e \in \mathrm{FE}_H(G)} g_{\mathrm{e}}(e)$$

for some edge-function $g_{\mathrm{e}} \colon V(G)^2 \to \mathbb{R}^+$.

Clique-sum-type [11]: f is of clique-sum-type if

$$f(G) = \min_{H \in \mathrm{MT}(G)} \sum_{\omega \in \mathrm{MC}(H)} g_{\mathrm{s}}(\omega)$$

for some *fast*[1] clique function $g_s\colon 2^{V(G)} \to \mathbb{R}^+$.

Given a graph parameter f and a graph G, a triangulation $H \in \mathrm{MT}(G)$ is *optimal* if (i) f is of clique-type with a clique function g_c and $f(G) = \max_{\omega\in\mathrm{MC}(H)} g_c(\omega)$, (ii) f is of fill-type with an edge function g_e and $f(G) = \sum_{e\in\mathrm{FE}_H(G)} g_e(e)$, or (iii) f is of clique-sum-type with a fast clique function g_s and $f(G) = \sum_{\omega\in\mathrm{MC}(H)} g_s(\omega)$.

Several well-known graph optimization problems are instantiations of one of these three types of graph parameters. In this work we evaluate the BT algorithm on the following five problems.

Treewidth [23, 27, 48]: The treewidth problem asks to compute

$$\mathrm{TW}(G) = \min_{H\in\mathrm{MT}(G)} \max_{\omega\in\mathrm{MC}(H)} \{|\omega| - 1\}.$$

The treewidth problem is of clique-type with $g_c(\omega) = |\omega| - 1$.

Minimum fill-in [23, 27, 46]: The minimum fill-in problem asks to compute

$$\mathrm{MF}(G) = \min_{H\in\mathrm{MT}(G)} \{|\mathrm{FE}_H(G)|\}.$$

The minimum fill-in problem is of fill-type with $g_e(e) = 1$.

Generalized & fractional hypertreewidth [31, 33, 45]. Hypergraphs generalize graphs by allowing arbitrary subsets of vertices as (hyper)edges. For a vertex $v \in V(\mathcal{G})$ of a hypergraph $\mathcal{G}$, let $E^v \subset E(\mathcal{G})$ be the set of edges containing v. A function $\gamma^K\colon E(\mathcal{G}) \to \{0,1\}$ ($\gamma^K\colon E(\mathcal{G}) \to [0,1]$) is an (a fractional) *edge cover* of a set $K \subset V(\mathcal{G})$ if $\sum_{e\in E^v} \gamma^K(e) \geq 1$ for each $v \in K$. The size of γ^K is $\sum_{e\in E(\mathcal{G})} \gamma^K(e)$. We denote the size of the smallest edge cover of K by $\mathrm{COV}_{\mathcal{G}}(K)$ and the size of the smallest fractional edge cover of K by $\mathrm{FCOV}_{\mathcal{G}}(K)$. The primal graph $\textsc{Prim}(\mathcal{G})$ of a hypergraph $\mathcal{G}$ has $V(\textsc{Prim}(\mathcal{G})) = V(\mathcal{G})$ and $E(\textsc{Prim}(\mathcal{G})) = \{\{u,v\} \mid \exists e \in E(\mathcal{G}), \{u,v\} \subset e\}$. Observe that each edge in $\textsc{Prim}(\mathcal{G})$ contains 2 vertices, i.e., all definitions from Section 2 are applicable.

The generalized hypertreewidth problem (GHTW) asks to compute

$$\mathrm{GHTW}(\mathcal{G}) = \min_{H\in\mathrm{MT}(\textsc{Prim}(\mathcal{G}))} \max_{\omega\in\mathrm{MC}(H)} \{\mathrm{COV}_{\mathcal{G}}(\omega)\}.$$

The fractional hypertreewidth problem (FHTW) asks to compute

$$\mathrm{FHTW}(\mathcal{G}) = \min_{H\in\mathrm{MT}(\textsc{Prim}(\mathcal{G}))} \max_{\omega\in\mathrm{MC}(H)} \{\mathrm{FCOV}_{\mathcal{G}}(\omega)\}.$$

Both problems are of clique-type with $g_c(\omega) = \mathrm{COV}_{\mathcal{G}}(\omega)$ and $g_c(\omega) = \mathrm{FCOV}_{\mathcal{G}}(\omega)$, respectively.

Total table size [11, 37, 41, 47]: Let G be a moralized Bayesian Network, i.e., an undirected graph, and $\textsc{ts}\colon V(G) \to \mathbb{N}$ a function, mapping each random variable $X \in V(G)$ to $\textsc{ts}(X)$, the size of the probability table associated with X in G. The total table size problem (TTS) asks to compute

$$\mathrm{TTS}(G) = \min_{H\in\mathrm{MT}(G)} \sum_{\omega\in\mathrm{MC}(H)} \left(\prod_{X\in\omega} \textsc{ts}(X)\right).$$

TTS is of clique-sum-type with $g_s(\omega) = \prod_{X\in\omega} \textsc{ts}(X)$. To see why g_s is fast, assume that $\textsc{ts}(X) \geq 2$ for all X and let $K_s \subset K \subset V(G)$. Then

$$g_s(K) = \prod_{X\in K} \textsc{ts}(X) = \prod_{X\in(K\setminus K_s)} \textsc{ts}(X) \prod_{X\in K_s} \textsc{ts}(X) \geq \prod_{X\in(K\setminus K_s)} 2 \prod_{X\in K_s} \textsc{ts}(X) \geq 2^{|K\setminus K_s|} g_s(K_s).$$

Note that the assumption $\textsc{ts}(X) \geq 2$ is not restrictive when considering random variables.

Finally, we note that an alternative way of defining the considered problems is through *tree decompositions* (see e.g. [10]). A tree decomposition of a graph G is a tuple (T, χ), where T is a tree

[1] A function g_s is fast if $g_s(K) \geq 2^{|K\setminus K_s|} g_s(K_s)$ for all $K_s \subset K$. The necessity of g_s being fast is discussed in [11].

and $\chi = \{X_t \mid t \in V(T)\}$ is a collection of subsets (bags) of the vertices of G, with the following properties.

(i) Each vertex of G is in some bag, i.e., $\cup_{t \in V(T)} X_t = V(G)$.
(ii) For each edge $e \in E(G)$, there is a bag $X \in \chi$ such that $e \subset X$.
(iii) For each vertex $v \in V(G)$, the set $\{t \in V(T) \mid v \in X_t\} \subset V(T)$ induces a connected subgraph of T.

Property (iii) is often referred to as the *running intersection property* and can be defined in several equivalent ways. One such alternative definition is $X_i \cap X_j \subset X_k$ for all nodes $i, j, k \in V(t)$ for which k lies on the (unique) path from i to j in T [9]. A (fractional) generalized hypertree decomposition of a hypergraph $\mathcal{G}$ is a tuple $((T^{\mathcal{G}}, \{X_1, \ldots, X_n\}), \{\lambda^1, \ldots, \lambda^n\})$, where $(T^{\mathcal{G}}, \{X_1, \ldots, X_n\})$ is a tree decomposition of $\textsc{Prim}(\mathcal{G})$ and λ^i is a (fractional) edge cover of X_i for all $i = 1, \ldots, n$. The alternative definitions of all optimization problems we consider are based on the bags of the tree decompositions. The equivalence between the definitions follows from the well-known connection between triangulations and tree decompositions [9, 29]. More precisely, given any triangulation H of G it is straightforward to obtain a tree decomposition of G with the maximal cliques of H as bags, and vice versa.

4 BT ALGORITHM

We continue with a description of the BT algorithm which our implementation of the approach is based on. Further details and formal justifications for BT algorithm can be found in several different sources [7, 11, 16, 17, 23, 27, 45].

4.1 Decomposing Graph Parameter Computation

Given a graph G and a graph parameter f, the BT algorithm decomposes the computation of $f(G)$ into the computation of $f(R(S, C))$ for all blocks (S, C). Recall that (S, C) is a block of G whenever $S \in \Delta(G)$ and C is a full component of S, i.e., $C \in \mathcal{C}(G \setminus S)$ and $\mathrm{N}(C) = S$. With slight abuse of notation, we also let $G = R(\emptyset, V(G))$. The correctness of the BT algorithm is due to the following theorem.

Theorem 4.1. *(Adapted from [27] and [11]) Let G be a graph and f one of the three types of graph parameters discussed in Section 3.*

If f is of clique-type with a clique function g_c, then

$$f(R(S,C)) = \min_{S \subsetneq \Omega \subset (S,C)} \max \left(g_c(\Omega), \max_{(S_i,C_i) \in (S,C:\Omega)} f(R(S_i,C_i)) \right).$$

If f is of fill-type with an edge function g_e, then

$$f(R(S,C)) = \min_{S \subsetneq \Omega \subset (S,C)} \left(\sum_{e \in \Omega^2 \setminus (E(G) \cup S^2)} g_e(e) + \sum_{(S_i,C_i) \in (S,C:\Omega)} f(R(S_i,C_i)) \right).$$

If f is of clique-sum-type with a fast clique function g_s, then

$$f(R(S,C)) = \min_{S \subsetneq \Omega \subset (S,C)} \left(g_s(\Omega) + \sum_{(S_i,C_i) \in (S,C:\Omega)} f(R(S_i,C_i)) \right).$$

In order to unify the three types of graph parameters in our implementation of the BT algorithm, we use two abstract functions *cliqueCost* and *mergeCost* which are instantiated depending on the graph parameter f being computed. The function *cliqueCost* computes the additional cost incurred by a PMC $\Omega \subset (S, C)$. To compute fill-type parameters, the function *cliqueCost* requires both Ω

and S as inputs. More concretely, whenever f is of clique-type or clique-sum-type with a (fast) clique function g_c, then $cliqueCost(\Omega, S) = g_c(\Omega)$. If f is of fill-type with an edge function g_e, then $cliqueCost(\Omega, S) = \sum_{e \in \Omega^2 \setminus (E(G) \cup S^2)} g_e(e)$. The function *mergeCost* merges the cost of the current Ω with the costs of the blocks in $(S, C : \Omega)$. More concretely, given a set X of numbers *mergeCost* combines them in a manner defined by f. If f is of clique-type, then $mergeCost(X) = \max(X)$ and if f is of fill-type or clique-sum-type, then $mergeCost(X) = \sum_{x \in X} x$. Using these two functions, Theorem 4.1 can be restated as follows.

Corollary 1. *Let G be a graph and f one of the three types of graph parameters discussed in Section 3. Then*

$$f(R(S,C)) = \min_{S \subsetneq \Omega \subset (S,C)} mergeCost\left(\{cliqueCost(\Omega, S)\} \cup \{f(R(S_i, C_i)) \mid (S_i, C_i) \in (S, C : \Omega)\}\right).$$

For a block (S, C) and a fixed PMC Ω satisfying $S \subsetneq \Omega \subset (S, C)$, the cost of $R(S, C)$ with respect to Ω is

$$mergeCost\left(\{cliqueCost(\Omega, S)\} \cup \{f(R(S_i, C_i)) \mid (S_i, C_i) \in (S, C : \Omega)\}\right).$$

Informally, the optimal cost of $R(S, C)$ with respect to Ω is computed by only considering triangulations of $R(S, C)$ in which Ω has been completed into a clique.

4.2 Detailed Description

Our implementation of the BT algorithm is presented in pseudocode as Algorithm 1. The implementation is mainly based on [7, 17, 23]. As mentioned, the BT algorithm works by decomposing the computation of $f(G) = f(R(\emptyset, V(G)))$ into the computation of $f(R(S, C))$ of all blocks of G. Furthermore, following Corollary 1, the value of $f(R(S, C))$ is computed as the minimum cost of $R(S, C)$ with respect to Ω over all $\Omega \in \Pi(G)$ satisfying $S \subsetneq \Omega \subset (S, C)$.

Algorithm 1 proceeds over triplets of form (Ω, S, C), where (S, C) is a block and $\Omega \in \Pi(G)$ satisfies $S \subsetneq \Omega \subset (S, C)$. The optimal cost of $R(S, C)$ with respect to Ω is computed on Lines 14–18. Whenever this cost is lower than the best known cost for $R(S, C)$ (Line 20), the value of $dp[(S, C)]$ is updated (Line 21) and Ω is stored in $optChoice[(S, C)]$ (Line 22). After processing all triplets (Lines 16–22), the value of each $dp[(S, C)]$ is equal to $f(R(S, C))$ for all blocks, and $optChoice[(S, C)]$ contains the PMC $\Omega \subset S \cup C$ that needs to be completed into a clique when constructing a optimal triangulation of $R(S, C)$. Specifically, the value of $f(G)$ is stored in $dp[(\emptyset, V(G))]$ (Line 25).

After running Algorithm 1, the optimal triangulation H can be reconstructed using a breadth-first search like procedure shown in Algorithm 2. Starting from $B = (\emptyset, V(G))$ (Line 3), the potential maximal clique Ω_B corresponding to $f(R(B))$ is completed into a clique (Line 7), and all blocks $B_i \in (B : \Omega_B)$ are added to the queue (Lines 8–10). Notice that $\Omega_B = optChoice[B]$.

We note that in order to compute the optimal cost of $R(S, C)$ for a PMC Ω, the optimal cost of all blocks in $(S, C : \Omega)$ needs to be computed. In our implementation of Algorithm 1, all triplets (Ω, S, C) are computed before the actual search (Lines 2–10), and then processed in order of increasing sizes of $S \cup C$ (Line 12). First all potential maximal cliques are enumerated using the procedure from [17] (Line 2) which in turn uses the procedure for enumerating all minimal separators from [7]. Then, for each potential maximal clique, all blocks (S, C) for which $S \subsetneq \Omega \subset (S, C)$ are initialized (Lines 4–10). The addition of the "dummy state" $(\Omega, \emptyset, V(G))$ on Line 10 is used for retrieving the optimal value $f(G)$.

We do not present pseudocode for enumerating potential maximal cliques here, as our implementation is directly based on the pseudocode of [17], using also the optimizations mentioned therein. Let G be a graph with n nodes, $v \in V(G)$ and $G' = G \setminus \{v\}$, i.e., G with the node v removed. The enumeration of potential maximal cliques is based on a characterization of $\Pi(G)$ in terms of $\Pi(G')$, $\Delta(G')$ and $\Delta(G)$. In other words, the set $\Pi(G) = \Pi(G[V_n])$ is computed by iteratively computing

ALGORITHM 1: BT algorithm

```
1  BT algorithm(G, cliqueCost, mergeCost)
2  Π ← EnumeratePMCs(G) [7, 17];
3  T ← {};
4  foreach Ω ∈ Π do
5      foreach D ∈ C(G \ Ω) do
6          S ← N(D);
7          C ← The component of G \ S such that Ω ⊂ S ∪ C;
8          T ← T ∪ {(Ω, S, C)};
9      end
10     T ← T ∪ {(Ω, ∅, V(G))};
11 end
12 sort T in increasing order of |S ∪ C| ;
13 dp[(S, C)] ← ∞ for all (S, C);
14 foreach (Ω, S, C) ∈ T do
15     cost ← cliqueCost(Ω, S);
16     foreach C′ ∈ C(G[C \ Ω]) do
17         S′ ← N(C′);
18         cost ← mergeCost(cost, dp[(S′, C′)]);
19     end
20     if cost < dp[(S, C)] then
21         dp[(S, C)] ← cost;
22         optChoice[(S, C)] ← Ω;
23     end
24 end
25 return dp[(∅, V(G))]
```

ALGORITHM 2: Reconstructing the optimal triangulation

```
1  Reconstruct(G, optChoice)
2  H ← G;
3  Q ← {({}, V(G))};
4  while Q not empty do
5      (S, C) ← pop(Q);
6      Ω ← optChoice[(S, C)];
7      E(H) ← E(H) ∪ Ω²;
8      foreach C′ ∈ C(G[C \ Ω]) do
9          S′ ← N(C′);
10         Q ← Q ∪ {(S′, C′)};
11     end
12 end
13 return H
```

$\Pi(G[V_i])$ for $i = 0, \ldots, n$ with $V_0 = \emptyset$ and $V_{i+1} = V_i \cup \{v_{i+1}\}$ for some $v_{i+1} \in V(G) \setminus V_i$. In our implementation the vertices are added to V_i in reverse order of the elimination order produced by maximum cardinality search [54]. The potential maximal clique enumeration algorithm uses the algorithm for minimal separator enumeration proposed in [7]. The set $\Delta(G)$ is computed by

extending the set of minimal separators included in the neighborhood $N(v)$ of each $v \in V(G)$ by a simple generation rule [7].

In order to analyze the total running time of Algorithm 1, we discuss the running time of the potential maximal clique enumeration (Line 2) and the DP search (Lines 4–22) separately. Let G be a graph with n vertices and m edges. The procedure for enumerating potential maximal cliques runs in polynomial time w.r.t the number of minimal separators, or more precisely, in $O(n^2m|\Delta(G)|^2)$ [17]. As $|\Pi(G)| \geq |\Delta(G)|/n$ [23], this is also polynomial time w.r.t the number of potential maximal cliques. The minimal separator enumeration algorithm used as a subroutine during potential maximal clique enumeration runs in $O(n^3|\Delta(G)|)$ time [7]. Since $|\Delta(G)| = O(1.6181^n)$ [25], the total running time of potential maximal clique enumeration is $O(2.6183^n)$.

The total running time of the of the DP search (Lines 4–22) depends on the graph parameter being computed, i.e., on the time complexities of the functions *cliqueCost* and *mergeCost*. For all of the problems we consider, each invocation of *mergeCost* is computable in constant time as $mergeCost(c_1, c_2)$ is either $c_1 + c_2$ or $\max(c_1, c_2)$. As the number of triplets (Ω, S, C) considered is $O(|\Pi(G)|n)$, the total time complexity of the DP search is $O(g_t(n)|\Pi(G)|n + |\Pi(G)|nm)$ where $g_t(n)$ is the time complexity of *cliqueCost*. For clique-type and clique-sum-type graph parameters, the value of $cliqueCost(\Omega, S)$ depends only on Ω. Hence the values can be precomputed, decreasing the total time complexity of the DP search to $O(g_t(n)|\Pi(G)| + |\Pi(G)|nm)$. For fill-type parameters we assume that the values $g_e(e)$ of the edge function can be computed in constant time for each edge e. The assumption is reasonable as the values can be precomputed for all $O(n^2)$ possible edges. This results in $O(|\Pi(G)|n^3)$ time complexity for the DP search on fill-type parameters. As $|\Pi(G)| = O(1.7549^n)$ [25], the total running time of the DP search with respect to n is $O(1.7549^n g_t(n))$. For an alternative view, if $cliqueCost(\Omega, S)$ is computable in polynomial time in n for each Ω and S the total running time of the DP search is $O(poly(n) \cdot |\Pi(G)|)$. For such problems, the time complexity of Algorithm 1 is dominated by the enumeration of potential maximal cliques, an observation that is supported by our experimental evaluation.

5 EXPERIMENTAL EVALUATION

We report on an evaluation of the empirical performance of Triangulator, our implementation of Algorithms 1 and 2, on the treewidth, minimum fill-in, generalized and fractional hypertreewidth, and total table size problems. For each problem, we compare the performance of Triangulator to available implementations of other previously-proposed algorithms for the individual problems. All experiments were run on computing nodes with dual 2.53-GHz Intel Xeon E5540 processors and 32 GB of main memory under Ubuntu Linux 14.04. Triangulator is available in open source under the MIT license at

https://github.com/Laakeri/Triangulator.

The benchmarks used in the evaluation reported on in this section are also available at this location. Before presenting detailed results for the individual problems, we provide further practical details on Triangulator.

5.1 Implementation Details

Triangulator is written in C++, with around 800 lines in the generic BT algorithm implementation and 4000 lines in total. We make use of the data structures in the C++ standard library (vector, set and map) to the extent possible and use recursive depth-first searches for standard graph traversal/modifying procedures. All additional data structures and algorithms were implemented from scratch. We note that careful low-level optimizations could further reduce the running time

of Triangulator by at least a reasonable constant factor. For further implementation-level details, the source code is available for inspection.

Before invoking Algorithm 1, Triangulator applies a number of preprocessing rules to the input graph G. First G is decomposed on its clique separators [53]. The value of $f(G[C_i])$ is computed separately for each component C_i in the decomposition and the value of $f(G)$ is obtained by merging the values of $f(G[C_i])$ for each component using the *mergeCost* function. Furthermore, the optimal triangulations of each component are merged to form the optimal triangulation of G. Computation of the clique separator decomposition of G is done by first computing a minimal triangulation $H \in \mathrm{MT}(G)$ [6, 50, 53]. The triangulation H is also used to obtain an upper bound U of $f(G)$. The bound allows instance-specific pruning of the set of potential maximal cliques that need to be considered during Algorithm 1. More specifically, any $\Omega \in \Pi(G)$ which is shown to not be a maximal clique in any optimal triangulation of G is ignored. For a simple example, if $f(G) = \mathrm{TW}(G)$, then all potential maximal cliques Ω for which $|\Omega| - 1 > U$ are ignored. We emphasize that these optimizations often do not significantly affect the overall running time of Triangulator. As discussed in the previous section, the overall running time of Triangulator is for most problems dominated by the enumeration of the PMCs. Since pruning is only done after a PMC has been computed, it does not decrease the total number of PMCs that are enumerated.

For treewidth and minimum fill-in, an additional preprocessing technique based on safe separators [14, 15] is applied. In more detail, let $v \in V(G)$ be a vertex of G such that the neighborhood $\mathrm{N}(v)$ requires one edge in order to be completed into a clique, i.e., such that $\mathrm{N}(v)^2 \setminus E(G[\mathrm{N}(v)]) = \{\{u, t\}\}$ for two vertices $u, t \in \mathrm{N}(v)$. During this phase of preprocessing, the edge $\{u, t\}$ is added to G [14, 15]. A special case of this rule in graphs containing no clique separators is the elimination of all degree-two vertices. Finally, for minimum fill-in, we also use a known kernelization algorithm [46]. Note that the kernelization algorithm also gives lower and upper bounds for minimum fill-in.

Instantiating *cliqueCost* is straightforward for most of the problems considered in this work. However, instantiating $\mathrm{COV}_{\mathcal{G}}(\omega)$ for the generalized hypertreewidth problem and $\mathrm{FCOV}_{\mathcal{G}}(\omega)$ for the fractional hypertreewidth problem is non-trivial. For computing $\mathrm{COV}_{\mathcal{G}}(\omega)$, we implemented a branch-and-bound set cover algorithm which branches on vertices that are still uncovered by the set of selected edges. When choosing which edge to cover a vertex with, the algorithm tries edges that cover a maximal number of uncovered vertices first. For computing $\mathrm{FCOV}_{\mathcal{G}}(\omega)$, we use the GLPK linear programming solver [42] and a straightforward linear programming model of fractional set cover.

5.2 Treewidth

Considering the treewidth problem, we compare the performance of Triangulator to those of the QuickBB branch-and-bound algorithm [30] and a declarative approach based on encoding treewidth as maximum satisfiability (MaxSAT) and applying a state-of-the-art MaxSAT solver to determine treewidth [3]. Furthermore, we give results for PIDDT [52], a so-called positive instance driven variant of BT algorithm which, employing problem-specific optimizations, took second place in the exact competition of the treewidth track of the 2017 PACE challenge.

Both QuickBB and the MaxSAT encoding make use of *elimination orderings* for computing the treewidth of a graph $G = (V, E)$. An elimination ordering of G is any linear ordering $\prec$ of V. A vertex $v_i \in V$ is a predecessor (in $\prec$) of another vertex v_j if $v_i \prec v_j$ and $\{v_i, v_j\} \in E$. The completion of G under $\prec$ is a directed graph, obtained by first adding edges between any two vertices v_i and v_j of G that have a common predecessor until fix-point, and then ordering all edges according to $\prec$. The width of $\prec$ is the maximum out-degree of any vertex in the completion of G under $\prec$ and the treewidth of G is the minimum width over all elimination orderings of G [10, 20].

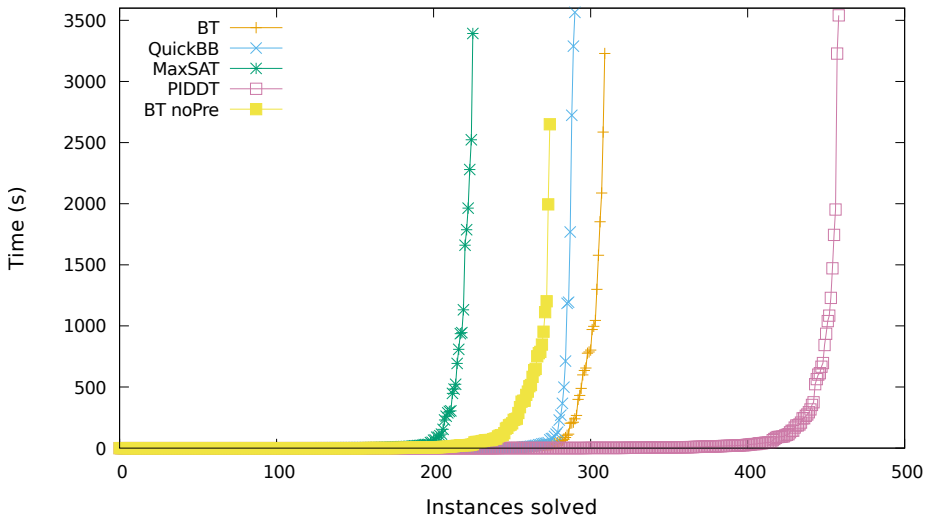

Fig. 2. Comparison of the empirical performance of Triangulator (BT), PIDDT, QuickBB, and MaxSAT on treewidth.

QuickBB searches over elimination orderings of the vertices of $G = (V, E)$. Each node in the search tree is a linear ordering $\prec^S$ of some subset $S \subset V$. The main observation underlying QuickBB is that the width of $\prec^S$ is a lower bound of the width of any elimination ordering $\prec$ obtained by completing $\prec^S$ to an elimination ordering of G. Thus a branch of the search tree can be pruned whenever the width of the ordering of that node is higher than the current known upper bound of the treewidth. QuickBB also uses a number of additional heuristics to prune the search space and limit the number of generated branches [30].

The MaxSAT encoding is based on encoding elimination orders via propositional constraints. Given a graph $G = (V, E)$, the encoding produces an unweighted MaxSAT instance $\mathcal{F}(G)$ with $|V| - 1$ unit-weighted soft propositional clauses $\{C_1, \ldots, C_{|V|-1}\}$ such that $\mathrm{TW}(G) \leq k$ if and only if there is a truth assignment that satisfies all hard clauses in $\mathcal{F}(G)$ and the soft clauses $C_k, \ldots, C_{|V|-1}$. The hard clauses of $\mathcal{F}(G)$ encode elimination orderings and, further, implement the completion procedure. We report results for the MaxSAT approach using the Maxino [1] MaxSAT solver. We note that Maxino performed the best on these instances out of the three best-performing solvers (namely Maxino, MaxHS [19] and Open-WBO [43]) in the unweighted track of the MaxSAT Evaluation 2017 [2].

PIDDT implements a variant of BT algorithm and is highly optimized for the decision version of the treewidth problem. More specifically, given a graph G and an integer k, PIDDT uses a dynamic programming scheme similar to Triangulator to decide if $\mathrm{TW}(G) \leq k$. In contrast to Triangulator, PIDDT uses treewidth-specific heuristics in order to enumerate only the PMCs required to decide if $\mathrm{TW}(G) \leq k$ during the dynamic programming search. Notice that computing the treewidth of G often requires iterating the algorithm for several different values of k. Following [52] we used a linear search strategy starting with k equal to the minimum degree of the graph and incrementing by 1 for each negative answer.

As benchmark instances for treewidth, we used instances from the PACE 2016 and 2017 treewidth and minimum fill-in challenges [21, 22], the DIMACS graphs available at http://mat.gsia.cmu.edu/

COLOR/instances.html, and Bayesian networks from the bnlearn Bayesian Network Repository (http://www.bnlearn.com/bnrepository/). Out of a total of 589 benchmarks obtained, 469 were solved by at least one of the three approaches within a per-instance timeout of 1 hour.

A comparison of the performance of Triangulator, QuickBB, MaxSAT, and PIDDT is presented in Figure 2 which shows the number of instances solved by each approach (x-axis) in terms of the per-instance time limit enforced on the algorithms (y-axis). Overall, within a per-instance time limit of 1 hour, Triangulator solved a total of 309 instances, whereas QuickBB solved 290 instances and Maxino on the MaxSAT encoding 225. We note that the MaxSAT approach specifically suffered from weak performance on the larger benchmarks, as the size of the declarative encoding increases. These results suggest that a generic implementation of the BT algorithm, such as Triangulator designed to be applicable to a range of problems, can exhibit better performance on the treewidth problem when compared to other specialized algorithms for the same problem. As for the impact of the preprocessing techniques applicable on treewidth, the "BT noPre" plot in Figure 2 shows the performance of Triangulator without applying preprocessing; we observed that preprocessing had a noticeable positive effect on runtime performance of some—but not on a majority of the—instances, and did not appear to significantly hurt performance on any of the instances. Furthermore, the even better performance of PIDDT, solving a total of 459 instances, shows the potential of applying problem-specific implementations within BT algorithm when aiming at solving a specific problem. However, rather than focusing on studying problem-specific optimizations, we continue by evaluating the generic approach of Triangulator on a variety of other problems.[2]

5.3 Minimum Fill-In

Considering the minimum fill-in problem, we compare the performance of Triangulator to that of a recent integer programming approach [4] which we will refer here to as IP. The IP approach is based on formulating the minimum fill-in problem as an integer program and applying problem-specific lazy constraint generation and a heuristic separation approach for obtaining cuts. The IP approach builds on earlier work by the same authors on a Benders decomposition approach [5] for minimum fill-in. We obtained the implementation of the IP approach directly from the authors, and, following the authors, used the IBM ILOG CPLEX integer programming solver [36] (version 12.7) as the underlying IP/LP solver.

An overview of the results for minimum fill-in is presented in Figure 3, using the same set of benchmarks as in our treewidth experiments. We note that there is a significant overlap between this benchmark set and the one used by the authors of the IP approach in their experiments [4]. Overall, Triangulator solved 299 instances, while the IP approach solved 228. Triangulator solved 10 more treewidth instances than minimum fill-in instances, even though the instances consists of exactly the same graphs. We hypothesize that this may be due to preprocessing being more effective on the treewidth objective function than on the minimum fill-in objective.

More detailed results, using the set of benchmarks used in evaluating the IP approach [4], are shown in Table 1. The faster running time for each instances given in boldface. Here we observe that Triangulator consistently outperforms the IP approach especially on the grid and queens families of benchmarks.

5.4 Generalized and Fractional Hypertreewidth

Turning to generalized and fractional hypertreewidth, we note that—to the best of our understanding—the only exact algorithm implementation for generalized hypertreewidth is BB-ghw [51]. For fractional hypertreewidth we are not aware of any available exact algorithm implementations.

[2]The optimizations of PIDDT are not directly applicable to the range of problems considered here.

Table 1. Running times of Triangulator (BT) and the IP approach to minimum fill-in for the instances used in [4]. Here $|V|$ and $|E|$ are the numbers of vertices and edges of the graph, respectively. The column Min-fill gives the cost of the optimal minimum fill-in as determined by the algorithms.

				Time (seconds)	
Instance	$\|V\|$	$\|E\|$	**Min-fill**	**BT**	**IP**
grid3_3	9	12	5	0.01	0.01
grid3_4	12	17	9	0.07	**0.02**
grid3_5	15	22	13	0.08	**0.04**
grid3_6	18	27	17	**0.01**	0.28
grid3_7	21	32	21	**0.01**	0.31
grid3_8	24	37	25	**0.03**	1.27
grid3_9	27	42	29	**0.03**	2.31
grid3_10	30	47	33	**0.16**	12.23
grid4_4	16	24	18	**0.01**	0.63
grid4_5	20	31	25	**0.04**	8.50
grid4_6	24	38	34	**0.16**	162.65
grid4_7	28	45	41	**0.59**	122.38
grid4_8	32	52	50	**2.33**	TO
grid4_9	36	59	57	**9.53**	3171.78
grid4_10	40	66	66	**37.28**	TO
grid5_5	25	40	37	**0.26**	59.08
grid5_6	30	49	50	**1.67**	TO
grid5_7	35	58	62	**10.61**	TO
grid5_8	40	67	75	**67.43**	TO
grid5_9	45	76	87	**416.76**	TO
grid6_6	36	60	69	**14.48**	TO
grid6_7	42	71	86	**149.75**	TO
grid7_7	49	84	111	**2760.10**	TO
queen3_3	9	28	5	0.01	0.01
queen3_4	12	46	12	0.01	0.01
queen3_5	15	67	22	**0.01**	0.02
queen3_6	18	91	36	**0.02**	0.09
queen3_7	21	118	53	**0.05**	0.61
queen3_8	24	148	74	**0.13**	2.05
queen3_9	27	181	98	**0.37**	6.11
queen3_10	30	217	126	**1.28**	29.25
queen4_4	16	76	26	**0.01**	0.02
queen4_5	20	110	51	**0.05**	0.65
queen4_6	24	148	83	**0.19**	7.21
queen4_7	28	190	119	**0.97**	44.00
queen4_8	32	236	164	**5.39**	353.85
queen4_9	36	286	217	**28.77**	TO
queen4_10	40	340	277	**152.82**	TO
queen5_5	25	160	93	**0.28**	16.14
queen5_6	30	215	144	**2.37**	134.12
queen5_7	35	275	214	**18.28**	TO
queen5_8	40	340	293	**158.52**	TO
queen5_9	45	410	386	**1367.10**	TO
queen5_10	50	485	-	TO	TO
queen6_6	36	290	231	**35.44**	TO
queen6_7	42	371	334	**509.27**	TO
queen6_8	48	458	-	TO	TO
queen6_9	54	551	-	TO	TO
queen6_10	60	650	-	TO	TO
queen7_7	49	476	-	TO	TO
queen7_8	56	588	-	TO	TO
queen7_9	63	707	-	TO	TO
queen7_10	70	833	-	TO	TO
queen8_8	64	728	-	TO	TO

				Time (seconds)	
Instance	$\|V\|$	$\|E\|$	**Min-fill**	**BT**	**IP**
anna	138	493	47	**0.15**	1.51
david	87	406	64	**1.39**	2.48
games120	120	638	-	TO	TO
huck	74	301	5	**0.01**	0.05
jean	77	254	16	**0.01**	0.08
miles250	125	387	53	**0.08**	2.63
miles500	128	1170	360	**2918.89**	TO
miles750	128	2113	471	2452.14	**459.00**
myciel3	11	20	10	0.01	0.01
myciel4	23	71	46	0.13	**0.02**
myciel5	47	236	196	119.06	**30.36**
1-FullIns_3	30	100	80	5.96	**5.06**
1-FullIns_4	93	593	-	TO	TO
1-Insertions_4	67	232	-	TO	TO
2-FullIns_3	52	201	-	TO	TO
2-Insertions_3	37	72	-	TO	TO
2-Insertions_4	149	541	-	TO	TO
3-FullIns_3	80	346	-	TO	TO
3-Insertions_3	56	110	-	TO	TO
4-FullIns_3	114	541	-	TO	TO
4-Insertions_3	79	156	-	TO	TO
DSJC125.1	125	736	-	TO	TO
DSJC125.5	125	3891	-	TO	TO
DSJC125.9	125	6961	726	**53.00**	TO
miles1000	128	3216	535	366.58	**282.09**
miles1500	128	5198	218	8.03	**3.21**
mug100_1	100	166	64	**0.02**	0.54
mug100_25	100	166	64	**0.02**	0.62
mug88_1	88	146	56	**0.01**	0.27
mug88_25	88	146	56	**0.01**	0.82
myciel6	95	755	-	TO	TO
r125.1	122	209	11	**0.01**	1.49
r125.1c	125	7501	207	**4.12**	48.22
r125.5	125	3838	1086	**537.45**	TO

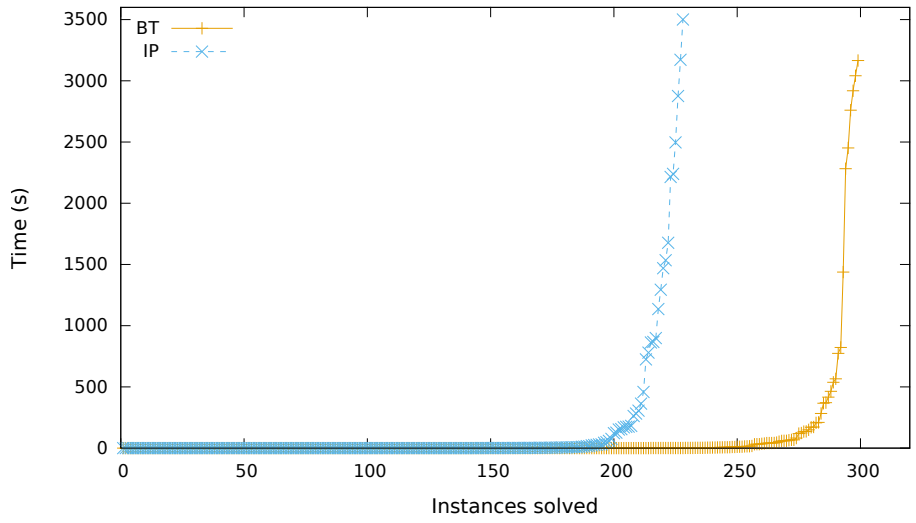

Fig. 3. Comparison of the empirical performance of different methods for minimum fill-in.

However, we will also report on the running times of the det-k-decomp [32] algorithm for deciding if a hypergraph has a (non-generalized, standard) hypertree decomposition of width at most k, thus solving a more restricted variants of the actual problems of generalized and fractional hypertreewidth.[3]

BB-ghw is an algorithm for computing the generalized hypertreewidth of a hypergraph $\mathcal{G}$ by branch-and-bound search over elimination orderings of the primal graph $\textsc{Prim}(\mathcal{G})$ of $\mathcal{G}$. Each node in the search tree corresponds to a partial elimination ordering of $\textsc{Prim}(\mathcal{G})$. The tree is pruned by computing a lower bound on $\mathrm{GHTW}(\mathcal{G})$ using the lower bound on $\mathrm{TW}(\textsc{Prim}(\mathcal{G}))$ and exactly solving an integer program.

A hypertree decomposition is a generalized hypertree decomposition $((T^{\mathcal{G}}, \{X_1, \ldots, X_n\}), \{\lambda^1, \ldots, \lambda^n\})$ in which each edge cover λ^i satisfies $\lambda^i(e) = 0$ for all edges e for which $e \setminus X_i \neq \emptyset$. The algorithm det-k-decomp decides whether a hypergraph has a (non-generalized) hypertree decomposition of width at most k by recursively branching on a separator of the graph. The hypertree decomposition is constructed recursively by branching on a separator of the graph. We use det-k-decomp to compute the hypertreewidth of a hypergraph $\mathcal{G}$ by linearly searching for the smallest k for which $\mathcal{G}$ has a hypertree decomposition of width at most k. For a fixed k, det-k-decomp runs in polynomial time, with the polynomial depending on k.

As benchmarks, we used the 265 hypertreewidth instances available at https://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html [28]. A per-instance time limit of 1 h was enforced on all approaches.

An overview of the results is presented in Figure 4, here using log scale for better readability. For generalized hypertreewidth, Triangulator solved 38 instances, while the directly comparable BB-ghw solved 27. Solving the fractional hypertreewidth problem appears more time-consuming than solving generalized hypertreewidth for Triangulator, with 33 instances solved for fractional treewidth. The det-k-decomp algorithm for (non-generalized) hypertreewidth solves 34 instances, less that Triangulator for generalized hypertreewidth.

[3] In general, $\mathrm{FHTW}(\mathcal{G}) \leq \mathrm{GHTW}(\mathcal{G}) \leq \mathrm{HTW}(\mathcal{G})$.

Table 2. Running times of Triangulator (BT) and BB-ghw for generalized hypertreewidth (GHTW), the BT algorithm for fractional hypertreewidth (FHTW), and det-k-decomp for (non-generalized) hypertreewidth (HTW). Here $|V|$ and $|E|$ are the numbers of vertices and edges of the hypergraph, respectively, and $|E'|$ is the number of edges in the primal graph. The columns GHTW, FHTW, and HTW give the actual widths for the individual instances as determined by the algorithms.

							Time (seconds)			
							GHTW		FHTW	HTW
Instance	$\|V\|$	$\|E\|$	$\|E'\|$	GHTW	FHTW	HTW	**BT**	**BB-ghw**	**BT**	**det-k-decomp**
adder_15	106	75	195	2	2.00	2	**0.01**	0.07	0.01	0.02
adder_25	176	125	325	2	2.00	2	**0.01**	0.09	0.02	0.03
adder_50	351	250	650	2	2.00	2	**0.01**	0.73	0.19	0.08
adder_75	526	375	975	2	2.00	2	**0.02**	1.41	0.11	0.17
adder_99	694	495	1287	2	2.00	2	**0.02**	TO	0.15	0.29
atv_partial_system	125	88	256	3	-	3	21.57	**0.07**	TO	1.97
b01	47	45	134	5	4.67	-	**3.58**	TO	36.76	TO
b02	27	26	72	3	3.00	3	0.06	**0.02**	0.69	0.03
b06	50	48	143	4	4.00	4	**1.39**	TO	12.18	38.35
bridge_15	137	135	285	2	-	2	**20.27**	TO	TO	0.03
bridge_25	227	225	475	2	-	2	**193.62**	TO	TO	0.06
bridge_50	452	450	950	-	-	2	TO	TO	TO	0.21
bridge_75	677	675	1425	-	-	2	TO	TO	TO	0.46
bridge_99	893	891	1881	-	-	2	TO	TO	TO	0.81
clique_10	45	10	360	5	5.00	5	**1.89**	TO	10.82	0.05
clique_15	105	15	1365	-	-	8	TO	TO	TO	52.25
dubois20	60	160	118	2	2.00	2	0.73	**0.07**	4.32	0.04
dubois21	63	168	124	2	2.00	2	0.54	**0.07**	2.90	0.04
dubois22	66	176	130	2	2.00	2	1.00	**0.09**	4.99	0.04
dubois23	69	184	136	2	2.00	2	1.06	**0.08**	5.68	0.04
dubois24	72	192	142	2	2.00	2	1.27	**0.11**	7.45	0.05
dubois25	75	200	148	2	2.00	2	1.50	**0.12**	7.78	0.05
dubois26	78	208	154	2	2.00	2	1.33	**0.12**	6.53	0.05
dubois27	81	216	160	2	2.00	2	1.53	**0.09**	7.80	0.06
dubois28	84	224	166	2	2.00	2	2.84	**0.19**	15.28	0.06
dubois29	87	232	172	2	2.00	2	3.31	**0.12**	19.95	0.06
dubois30	90	240	178	2	2.00	2	3.26	**0.19**	15.81	0.07
dubois50	150	400	298	2	2.00	2	23.79	**0.44**	87.55	0.16
dubois100	300	800	598	2	-	2	767.99	**0.74**	TO	0.58
grid2d_10	50	50	161	4	-	4	**413.40**	TO	TO	41.51
grid3d_4	32	32	156	5	4.50	5	**6.22**	TO	123.42	424.74
grid4d_3	41	40	272	6	-	-	**1276.95**	TO	TO	TO
grid5	25	40	40	3	3.00	3	**0.38**	242.31	3.03	0.08
hole6	42	133	231	7	-	-	48.27	**0.20**	TO	TO
hole7	56	204	364	8	-	-	1695.22	**0.55**	TO	TO
hole8	72	297	540	9	-	-	TO	**1.19**	TO	TO
ii8a1	66	186	447	7	-	-	**1156.25**	TO	TO	TO
jnh1	100	850	4391	-	11.14	-	TO	TO	596.94	TO
jnh201	100	800	4323	-	11.16	-	TO	TO	795.68	TO
jnh301	100	900	4450	-	11.16	-	TO	TO	467.89	TO
jnh305	100	900	4374	-	11.36	-	TO	TO	670.09	TO
jnh310	100	900	4358	-	11.46	-	TO	TO	737.49	TO
NewSystem1	142	83	329	3	3.00	3	**3.51**	TO	18.50	2.07
NewSystem2	345	198	799	3	-	3	**155.15**	TO	TO	58.54
s27	17	13	29	2	2.00	2	0.01	0.01	0.01	0.01
uf20-01	20	91	147	6	5.33	-	**0.55**	1.44	0.09	TO
uf20-050	20	91	158	6	5.67	-	**0.25**	0.38	0.06	TO
uf20-099	20	91	147	6	5.33	-	**0.25**	2.08	0.08	TO

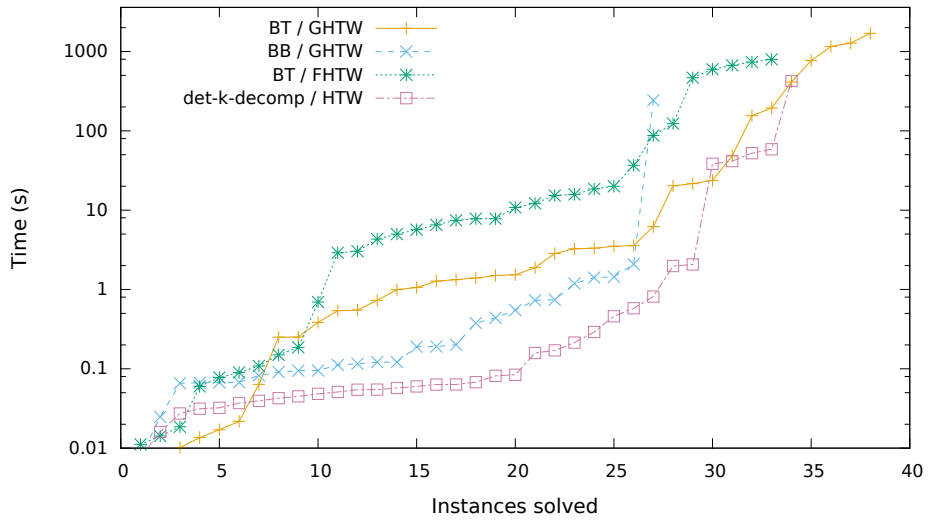

Fig. 4. Comparison of approaches for hypergraph measures.

A more detailed listing of the results for each instance solved by at least one of the approaches (as the respective problem variant) is presented in Table 2, with the faster running time of Triangulator and BB-ghw reported in boldface for each instance for the generalized hypertreewidth problem. We observe that Triangulator is consistently competitive with BB-ghw; BB-ghw is faster mainly on instances on which Triangulator also takes only a few seconds at most.

5.5 Total Table Size

Finally, considering total table size, we compare the performance of Triangulator with that of the recently proposed extended depth-first search (EDFS) algorithm [41]. EDFS is an extension of a depth-first search algorithm [47] based on a branch-and-bound search over vertex elimination orderings while efficiently maintaining the set of maximal cliques of a partially triangulated graph. The set of maximal cliques is maintained for computing the total table size of the partial solution, which is then used for pruning the search space. The EDFS algorithm improves the basic depth-first search algorithm by introducing a new maximal clique maintenance algorithm and a new rule for pruning elimination orderings that lead to identical triangulations.

As total table size benchmarks, we used Bayesian networks from the standard bnlearn Bayesian Network Repository, following [41]. Version 8 of the Java Virtual Machine was used for running the EDFS algorithm. A per-instances time limit of 2 hours was enforced on the algorithms.

Detailed results are presented in Table 3. We observe that Triangulator is clearly competitive, exhibiting faster running times for each instance solved by the approaches.

6 CONCLUSIONS

The BT algorithm originally proposed by Bouchitté and Todinca for the treewidth and minimum fill-in problems yields good exponential-time algorithms for various graph optimization problems where the underlying goal is to find an optimal graph triangulation (under different notions of optimality). While there has been noticeable interest in extensions and variations of the BT algorithm in terms of theoretical analysis, there are few reported works on the implementation and empirical

Table 3. Running times of Triangulator (BT) and EDFS for the total table size problem. Here $|V|$ and $|E|$ are the number of vertices and edges, respectively, of the moralized Bayesian network. The column TTS gives the total table size as determined by the algorithms.

				Time (seconds)	
Instance	$\|V\|$	$\|E\|$	**TTS**	**BT**	**EDFS**
alarm	37	65	996	**0.01**	0.35
andes	223	626	-	TO	TO
barley	48	126	17140796	**2.25**	4458.14
child	20	30	642	**0.01**	0.31
diabetes	413	819	-	TO	TO
hailfinder	56	99	9406	**0.08**	3.28
hepar2	70	158	2617	**0.01**	0.42
insurance	27	70	23880	**0.03**	0.83
mildew	35	80	3400464	**0.51**	2.95
munin1	186	354	-	TO	TO
pathfinder	109	208	182641	**0.08**	3.04
pigs	441	806	-	TO	TO
water	32	123	3028305	**0.15**	2.80
win95pts	76	225	2684	**0.22**	10.88

evaluation of this class of algorithms. To bridge this gap, in this paper we empirically evaluated our implementation of the BT algorithm for five well-known problems the algorithm can be extended to. For each of the problems, we provided an empirical comparison of our open source BT implementation called Triangulator with other algorithmic approaches proposed and implemented for the distinct problem. The empirical results suggest that implementing and extending the BT algorithm yields an empirically competitive approach to each of the considered problems. Improvements to the procedure for enumerating potential maximal cliques as a current performance bottleneck can be expected to yield further improvements in terms of empirical running times, and is thereby an interesting direction for further work. The empirical results on the five problems considered in this work also suggest that the BT algorithm could provide a practical solution method for futher problems, for example by studying ways of making use of further extensions of the approach [24] in practice.

ACKNOWLEDGMENTS

This work has been financially supported by Academy of Finland (grants 251170 COIN, 276412, 284591, and 312662) and the DoCS Doctoral Programme in Computer Science and the Research Funds of the University of Helsinki.

REFERENCES

[1] Mario Alviano, Carmine Dodaro, and Francesco Ricca. 2015. A MaxSAT algorithm using cardinality constraints of bounded size. In *Proc. AAAI*. AAAI Press, 2677–2683.

[2] Carlos Ansótegui, Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. 2017. MaxSAT Evaluation 2017. http://mse17.cs.helsinki.fi/.

[3] Jeremias Berg and Matti Järvisalo. 2014. SAT-based approaches to treewidth computation: an evaluation. In *Proc. ICTAI*. IEEE Computer Society, 328–335.

[4] David Bergman, Carlos H. Cardonha, André Augusto Ciré, and Arvind U. Raghunathan. 2016. On the minimum chordal completion polytope. *CoRR* abs/1612.01966 (2016). http://arxiv.org/abs/1612.01966

[5] David Bergman and Arvind U. Raghunathan. 2015. A Benders approach to the minimum chordal completion problem. In *Proc. CPAIOR (Lecture Notes in Computer Science)*, Vol. 9075. Springer, 47–64.

[6] Anne Berry, Jean R. S. Blair, Pinar Heggernes, and Barry W. Peyton. 2004. Maximum Cardinality Search for Computing Minimal Triangulations of Graphs. *Algorithmica* 39, 4 (2004), 287–298.

[7] Anne Berry, Jean-Paul Bordat, and Olivier Cogis. 1999. Generating all the minimal separators of a graph. In *Proc. WG (Lecture Notes in Computer Science)*, Vol. 1665. Springer, 167–172.

[8] Umberto Bertele and Francesco Brioschi. 1972. *Nonserial Dynamic Programming*. Academic Press, Inc., Orlando, FL, USA.

[9] Hans L Bodlaender. 1998. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science* 209, 1 (1998), 1–45.

[10] Hans L. Bodlaender. 2005. Discovering treewidth. In *Proc. SOFSEM (Lecture Notes in Computer Science)*, Vol. 3381. Springer, 1–16.

[11] Hans L. Bodlaender and Fedor V. Fomin. 2005. Tree decompositions with small cost. *Discrete Applied Mathematics* 145, 2 (2005), 143–154.

[12] Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. 2012. On exact algorithms for treewidth. *ACM Transactions on Algorithms* 9, 1, Article 12 (2012).

[13] Hans L Bodlaender, John R Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. 1995. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms* 18, 2 (1995), 238–255.

[14] Hans L. Bodlaender, Pinar Heggernes, and Yngve Villanger. [n. d.]. Faster Parameterized Algorithms for Minimum Fill-in. *Algorithmica* 61, 4 ([n. d.]), 817–838.

[15] Hans L. Bodlaender and Arie M.C.A. Koster. 2006. Safe separators for treewidth. *Discrete Mathematics* 306, 3 (2006), 337–350.

[16] Vincent Bouchitté and Ioan Todinca. 2001. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.* 31, 1 (2001), 212–232.

[17] Vincent Bouchitté and Ioan Todinca. 2002. Listing all potential maximal cliques of a graph. *Theoretical Computer Science* 276, 1 (2002), 17–32.

[18] Gregory F. Cooper. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42, 2-3 (1990), 393 – 405.

[19] Jessica Davies and Fahiem Bacchus. 2013. Exploiting the power of MIP solvers in MaxSAT. In *Proc. SAT (Lecture Notes in Computer Science)*, Vol. 7962. Springer, 166–181.

[20] Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113, 1-2 (1999), 41–85.

[21] Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. 2016. The First Parameterized Algorithms and Computational Experiments Challenge. In *Proc. IPEC (LIPIcs)*, Vol. 63. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 30:1–30:9.

[22] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. 2018. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The second iteration. In *Proc. IPEC 2017 (LIPIcs)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 30:1–30:12.

[23] Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. 2008. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.* 38, 3 (2008), 1058–1079.

[24] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. 2015. Large Induced Subgraphs via Triangulations and CMSO. *SIAM J. Comput.* 44, 1 (2015), 54–87. https://doi.org/10.1137/140964801

[25] Fedor V. Fomin and Yngve Villanger. 2008. Treewidth computation and extremal combinatorics. In *Proc. ICALP (Lecture Notes in Computer Science)*, Vol. 5125. Springer, 210–221.

[26] Eugene C. Freuder. 1990. Complexity of K-tree structured constraint satisfaction problems. In *Proc. AAAI*. AAAI Press, 4–9.

[27] Masanobu Furuse and Koichi Yamazaki. 2014. A revisit of the scheme for computing treewidth and minimum fill-in. *Theoretical Computer Science* 531 (2014), 66–76.

[28] Tobias Ganzow, Georg Gottlob, Nysret Musliu, and Marko Samer. 2005. *A CSP hypergraph library*. Technical Report DBAI-TR-2005-50. Vienna University of Technology Institute of Information Systems (DBAI).

[29] Fănică Gavril. 1974. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B* 16, 1 (1974), 47–56.

[30] Vibhav Gogate and Rina Dechter. 2004. A complete anytime algorithm for treewidth. In *Proc. UAI*. AUAI Press, 201–208.

[31] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree decompositions and tractable queries. *J. Comput. System Sci.* 64, 3 (2002), 579–627.

[32] Georg Gottlob and Marko Samer. 2009. A backtracking-based algorithm for hypertree decomposition. *Journal of Experimental Algorithmics* 13, Article 1 (2009).

[33] Martin Grohe and Dániel Marx. 2014. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms* 11, 1, Article 4 (2014).

[34] Rob Gysel. 2014. Minimal triangulation algorithms for perfect phylogeny problems. In *Proc. LATA (Lecture Notes in Computer Science)*, Vol. 8370. Springer, 421–432.

[35] Rob Gysel, Kristian Stevens, and Dan Gusfield. 2012. Reducing problems in unrooted tree compatibility to restricted triangulations of intersection graphs. In *Proc. WABI (Lecture Notes in Computer Science)*, Vol. 7534. Springer, 93–105.
[36] IBM ILOG. 2017. CPLEX optimizer 12.7.0. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer
[37] Finn V. Jensen and Frank Jensen. 1994. Optimal junction trees. In *Proc. UAI*. Morgan Kaufmann Publishers Inc., 360–366.
[38] Sampath K. Kannan and Tandy J. Warnow. 1994. Inferring evolutionary history from DNA sequences. *SIAM J. Comput.* 23, 4 (1994), 713–737.
[39] Uffe Kjaerulff. 1990. *Triangulation of graphs — algorithms giving small total state space.* Research Report R-90-09. Department of Mathematics and Computer Science, Aalborg University, Denmark.
[40] Steffen L. Lauritzen and David J. Spiegelhalter. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)* 50, 2 (1988), 157–194.
[41] Chao Li and Maomi Ueno. 2017. An extended depth-first search algorithm for optimal triangulation of Bayesian networks. *International Journal of Approximate Reasoning* 80 (2017), 294–312.
[42] A Makhorin. 2001. GLPK-the GNU linear programming toolkit. https://www.gnu.org/software/glpk/
[43] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. 2014. Open-WBO: A modular MaxSAT solver. In *Proc. SAT (Lecture Notes in Computer Science)*, Vol. 8561. Springer, 438–445.
[44] Dániel Marx. 2010. Approximating fractional hypertree width. *ACM Transactions on Algorithms* 6, 2, Article 29 (2010), 17 pages.
[45] Lukas Moll, Siamak Tazari, and Marc Thurley. 2012. Computing hypergraph width measures exactly. *Inform. Process. Lett.* 112, 6 (2012), 238–242.
[46] Assaf Natanzon, Ron Shamir, and Roded Sharan. 2000. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Comput.* 30, 4 (2000), 1067–1079.
[47] Thorsten J. Ottosen and Jirí Vomlel. 2012. All roads lead to Rome—New search methods for the optimal triangulation problem. *International Journal of Approximate Reasoning* 53, 9 (2012), 1350–1366.
[48] Neil Robertson and Paul D. Seymour. 1986. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms* 7, 3 (1986), 309–322.
[49] Donald J. Rose. 1970. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.* 32, 3 (1970), 597–609.
[50] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. 1976. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM J. Comput.* 5, 2 (1976), 266–283.
[51] Werner Schafhauser. 2006. *New heuristic methods for tree decompositions and generalized hypertree decompositions.* Master's thesis. Vienna University of Technology.
[52] Hisao Tamaki. 2017. Positive-instance driven dynamic programming for treewidth. In *Proc. ESA (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 87. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 68:1–68:13.
[53] Robert E. Tarjan. 1985. Decomposition by clique separators. *Discrete Mathematics* 55, 2 (1985), 221–232.
[54] Robert E. Tarjan and Mihalis Yannakakis. 1984. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.* 13, 3 (1984), 566–579.

Received March 2018; revised October 2018; accepted December 2018